

Ejercicio1:

1.

Python Password Validation Function

Haz una función en Python para validar una contraseña.

Te ayudo a crear una función para validar contraseñas en Python. Aquí tienes una implementación completa con diferentes niveles de validación:

Validador de Contraseñas

Código

He creado un validador de contraseñas completo en Python con las siguientes características:

Funciones principales:

- validar_contraseña()** - Valida según criterios específicos por nivel
- evaluar_fortaleza()** - Evalúa y puntúa la fortaleza de la contraseña
- generar_sugerencias()** - Proporciona consejos para mejorar la contraseña
- analizar_contraseña()** - Función principal que combina todo el análisis

Niveles de validación:

- Básico:** Mínimo 8 caracteres, mayúscula, minúsculas y números

Responder a Claude...

Copiar Publicar

```
def validar_contraseña(pwd, nivel="básico"):
    """Valida una contraseña según el nivel especificado.
    Retorna True si cumple con los requisitos, False en caso contrario.
    """
    # Reglas básicas
    if len(pwd) < 8:
        return False
    if not re.match(r'[a-zA-Z0-9]', pwd):
        return False

    # Reglas avanzadas
    if nivel == "avanzado":
        if not re.match(r'[a-zA-Z0-9!@#$%^&*]', pwd):
            return False
        if not re.match(r'[a-zA-Z]', pwd):
            return False
        if not re.match(r'[0-9]', pwd):
            return False

    return True

# Ejemplos de uso
contrasenas_ejemplo = [
    "P@ssw0rd123",
    "12345678",
    "qwertyuiop",
    "1qaz!@WSX",
    "1234567890"
]

print("EJEMPLOS DE VALIDACIÓN DE CONTRASEÑAS")
print("="*60)

for i, pwd in enumerate(contrasenas_ejemplo, 1):
    print(f"\nEjemplo {i}: '{pwd}'")
    analizar_contraseña(pwd, "medio")

# Ejemplo interactivo
print("\n" + "="*60)
print("PRUEBA TU PROPIA CONTRASEÑA")
print("="*60)

try:
    mi_contraseña = input("\nIngresa una contraseña para analizar: ")
    nivel_elegido = input("Elige el nivel (basico/medio/alto) [medio]: ").lower()

    if nivel_elegido not in ["basico", "medio", "alto"]:
        nivel_elegido = "medio"
        print(f"Nivel no válido, usando 'medio'")

    analizar_contraseña(mi_contraseña, nivel_elegido)

except KeyboardInterrupt:
    print("\n¡Hasta luego!")
except Exception as e:
    print(f"\nError: {e}")
```

2.

Python Password Validation Function

Rol Eres un desarrollador de software senior especializado en ciberseguridad.

Tarea: Genera una función en Python llamada `validar_contraseña`.

Contexto: La función recibirá una cadena de texto (la contraseña) y debe retornar `True` si cumple con TODAS las siguientes reglas, o `False` si falla al menos una.

Reglas de Validación (Criterios):

- Debe tener una longitud mínima de 12 caracteres.
- Debe contener al menos una letra mayúscula (A-Z).
- Debe contener al menos una letra minúscula (a-z).
- Debe contener al menos un número (0-9).
- Debe contener al menos un carácter especial (ej: @, #, \$, %, &, !).

Formato de Salida:

- El bloque de código de la función en Python.
- Debe incluir docstrings que expliquen qué hace la función, sus parámetros y qué retorna.
- Añade 3 ejemplos de uso: uno con una contraseña válida y dos con contraseñas inválidas que demuestren diferentes fallos.

Restricciones:

- No uses librerías externas, solo el core de Python.
- El código debe ser legible y seguir las convenciones de estilo de PEP 8.

Función de Validación de Contraseñas

Código

He creado la función `validar_contraseña` que cumple con todos los requisitos especificados. La función implementa una validación robusta de contraseñas siguiendo las

Responder a Claude...

Copiar Publicar

```
def validar_contraseña(pwd):
    """Valida una contraseña según los criterios de seguridad.
    Retorna True si cumple con todos los requisitos, False en caso contrario.
    """
    # Reglas de validación
    if len(pwd) < 12:
        return False
    if not re.match(r'[A-Z]', pwd):
        return False
    if not re.match(r'[a-z]', pwd):
        return False
    if not re.match(r'[0-9]', pwd):
        return False
    if not re.match(r'[!@#$%^&*]', pwd):
        return False

    return True

# Ejemplo 1: Contraseña válida
contrasena_valida = "P@ssw0rd123!"
resultado1 = validar_contraseña(contrasena_valida)
print(f"Contraseña: '{contrasena_valida}' -> Válida: {resultado1}")

# Ejemplo 2: Contraseña inválida - Sin carácter especial
contrasena_sin_especial = "MiContraseña123"
resultado2 = validar_contraseña(contrasena_sin_especial)
print(f"Contraseña: '{contrasena_sin_especial}' -> Válida: {resultado2}")

# Ejemplo 3: Contraseña inválida - Sin longitud suficiente
contrasena_corta = "Pass123@"
resultado3 = validar_contraseña(contrasena_corta)
print(f"Contraseña: '{contrasena_corta}' -> Válida: {resultado3}")

print("\n--- Análisis detallado ---")

# Función auxiliar para mostrar análisis detallado
def analizar_contraseña(contrasena):
    print(f"\nAnálisis de: '{contrasena}'")
    print(f"Longitud: {len(contrasena)} caracteres ({'✓' if len(contrasena) >= 12 else '✗'})")

    tiene_mayuscula = any(c.isupper() for c in contrasena)
    tiene_minuscula = any(c.islower() for c in contrasena)
    tiene_numero = any(c.isdigit() for c in contrasena)
    tiene_especial = any(c in "@#$%&*" for c in contrasena)

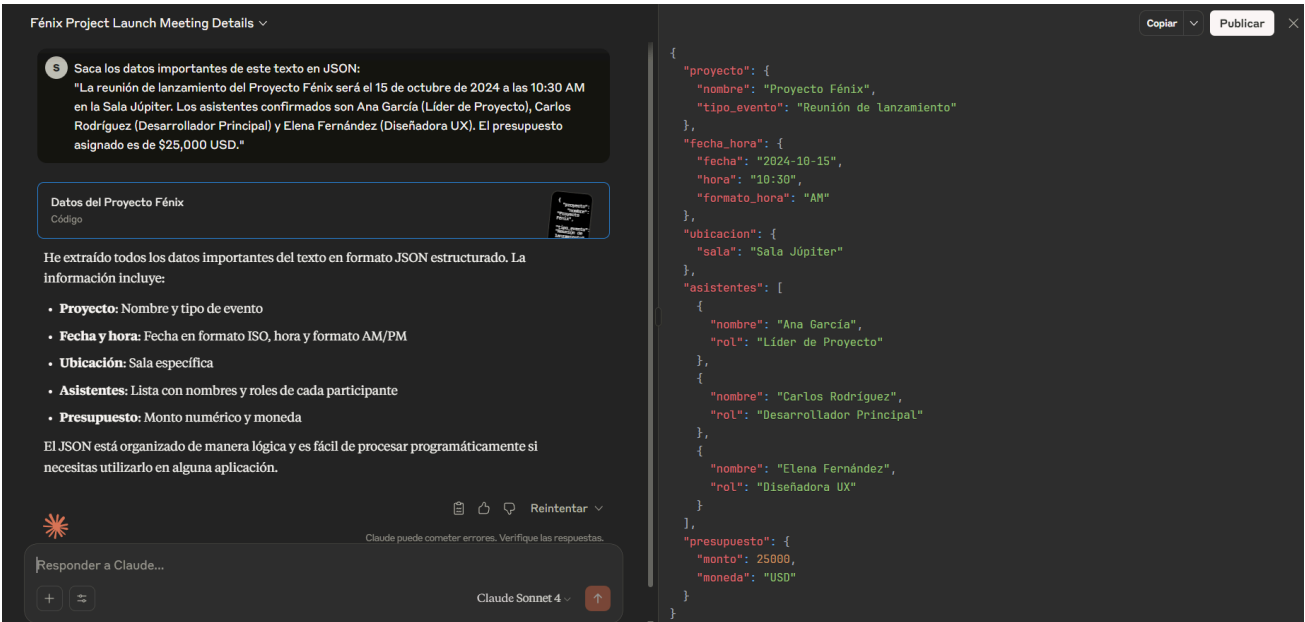
    print(f"Mayúscula: {'✓' if tiene_mayuscula else '✗'}")
    print(f"Minúscula: {'✓' if tiene_minuscula else '✗'}")
    print(f"Número: {'✓' if tiene_numero else '✗'}")
    print(f"Carácter especial: {'✓' if tiene_especial else '✗'}")
    print(f"Resultado: {'VÁLIDA' if validar_contraseña(contrasena) else 'INVÁLIDA'}")

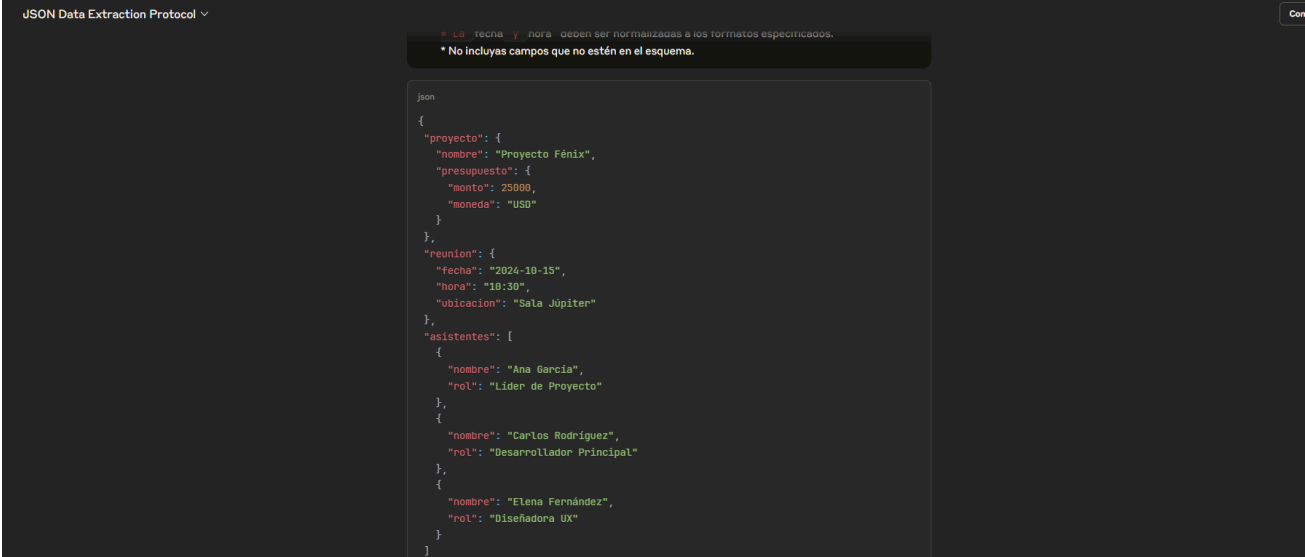
# Análisis detallado de los ejemplos
analizar_contraseña(contrasena_valida)
analizar_contraseña(contrasena_corta)
analizar_contraseña(contrasena_sin_especial)
```

- **Complejidad:** ¿La función del prompt simple incluyó todas las reglas?
R// No, intento incluir algunas y de alguna manera lograr clasificar la contraseña según si cumplía ciertas reglas pero no todas las reglas que por ejemplo son definidas en el prompt avanzado.
- **Claridad:** ¿El código está bien documentado? ¿Es fácil de entender?
R// Si, en ambos casos el LLM documentó muy bien el código que generó.
- **Utilidad:** ¿El prompt avanzado proporcionó ejemplos de uso que facilitan la prueba y la integración?
R// Si, proporciono 3 ejemplos que facilitan la integración.

- **Robustez:** ¿La función maneja casos borde que el prompt simple podría haber ignorado?
R// Si.

Ejercicio 2:

1. 

2. 

```
json
{
  "proyecto": {
    "nombre": "Proyecto Fénix",
    "presupuesto": {
      "monto": 25000,
      "moneda": "USD"
    }
  },
  "reunion": {
    "fecha": "2024-10-15",
    "hora": "10:30",
    "ubicacion": "Sala Júpiter"
  },
  "asistentes": [
    {
      "nombre": "Ana García",
      "rol": "Líder de Proyecto"
    },
    {
      "nombre": "Carlos Rodríguez",
      "rol": "Desarrollador Principal"
    },
    {
      "nombre": "Elena Fernández",
      "rol": "Diseñadora UX"
    }
  ]
}
```

2.C: Evaluación Comparativa

- **Estructura:** ¿El JSON del prompt simple tiene una estructura lógica y predecible?
R// Lógica si, predecible si no.
- **Precisión:** ¿Los tipos de datos son correctos (números vs. strings)?
R// Si, en ambos son correctos, incluso con el formato de fecha.
- **Parseabilidad:** ¿El JSON generado por el prompt avanzado es directamente utilizable por una aplicación sin necesidad de limpieza o conversión?

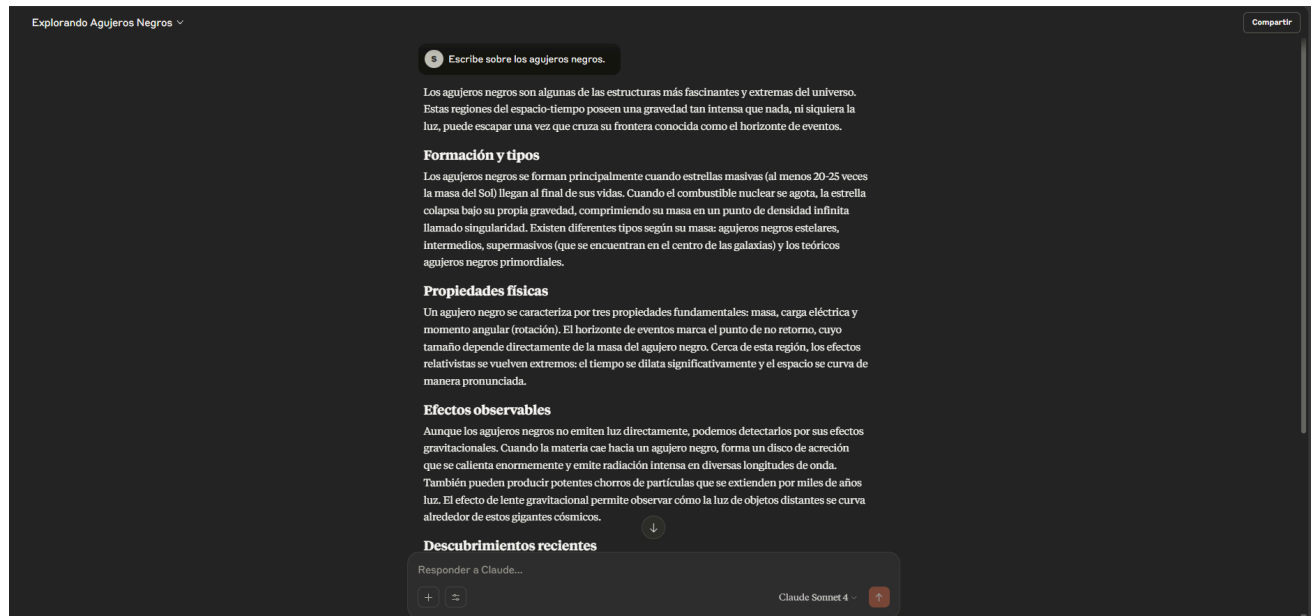
R//Si.

- **Manejo de Nulos:** ¿Cómo manejaría cada uno un texto donde, por ejemplo, el presupuesto no se menciona?

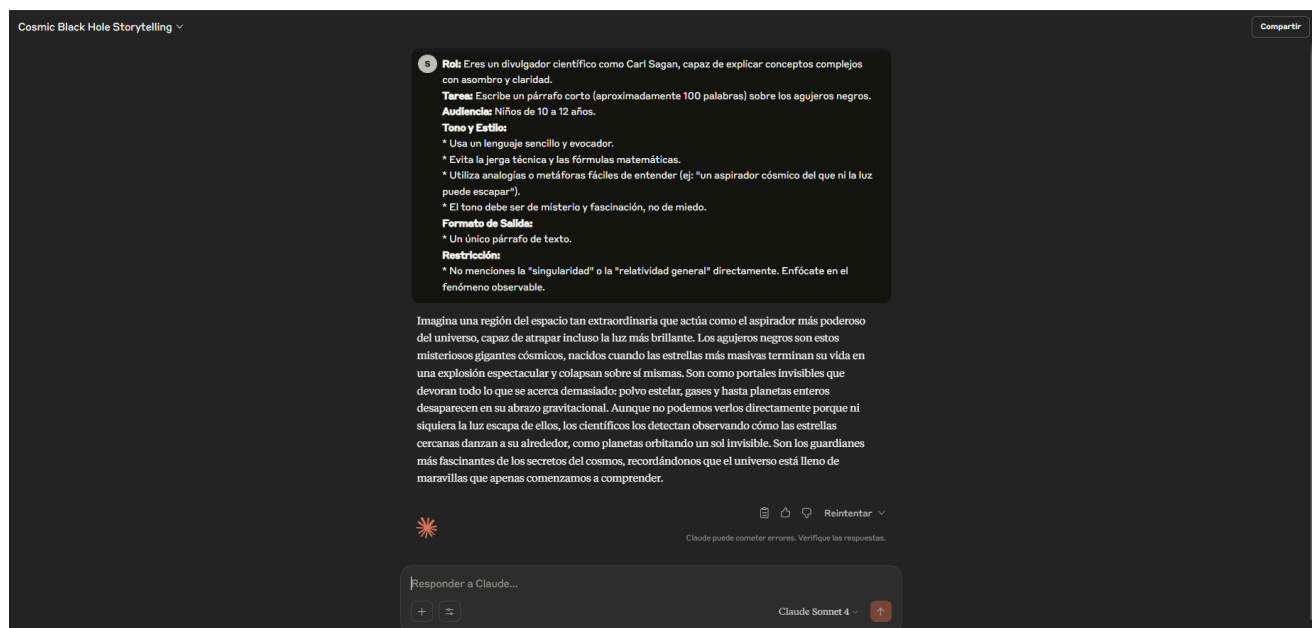
R// En el prompt simplificado no lo incluiría dentro del json debido a que directamente no lo encontraría como información y en el prompt Avanzado lo pondría como null.

Ejercicio 3:

1.



2.



3.C: Evaluación Comparativa

- **Adecuación a la Audiencia:** ¿Qué respuesta es más apropiada y comprensible para un niño?

R// En su mayoría si, aunque el niño quedaria curioso de varias cosas.

- **Tono:** ¿Logró el prompt simple capturar el tono de asombro solicitado?
R// En ciertos puntos del texto que se generó sí, pero no es constante.
- **Creatividad:** ¿Qué respuesta es más original y memorable?
R// La segunda, pues no escribe sobre el tema como para cualquiera sino que busca una forma original para hacerlo entendible para niños.
- **Cumplimiento de Restricciones:** ¿El prompt avanzado logró evitar la jerga técnica como se le pidió?
R// Sí.